SQL: Part I

CPS 116
Introduction to Database Systems

Announcements (Tue. Sep. 20)

❖ Homework #1 due today
  • Deadline for the Gradiance portion has been extended
❖ Homework #2 will be assigned Thursday

SQL

❖ SQL: Structured Query Language
  • Pronounced “S-Q-L” or “sequel”
  • The standard query language supported by most commercial DBMS
❖ A brief history
  • IBM System R
  • ANSI SQL89
  • ANSI SQL92 (SQL2)
  • ANSI SQL99 (SQL3)
  • ANSI SQL 2003 (added OLAP, XML, etc.)
  • ANSI SQL 2006 (added more XML)
  • ANSI SQL 2008, …
Creating and dropping tables

- **CREATE TABLE** `table_name` (
  `...`, `column_name` `column_type`, ...
);
- **DROP TABLE** `table_name`;

**Examples**

```sql
create table Student (SID integer,
  name varchar(30),
  email varchar(30),
  age integer, GPA float);
create table Course (CID char(10),
  title varchar(100));
create table Enroll (SID integer,
  CID char(10));
drop table Student;
drop table Course;
drop table Enroll;
```

-- everything from -- to the end of the line is ignored.
-- SQL is insensitive to white space.
-- SQL is insensitive to case (e.g., ...Course... is equivalent to
-- ...COURSE...)

Basic queries: SFW statement

- **SELECT** `A_1`, `A_2`, ..., `A_n`
  FROM `R_1`, `R_2`, ..., `R_m`
  WHERE `condition`;
- Also called an SPJ (select-project-join) query
- Equivalent (not really!) to relational algebra query
  \[ \pi_{A_1, A_2, ..., A_n}(\sigma_{condition}(R_1 \times R_2 \times ...) \times R_m)) \]

Example: reading a table

- **SELECT** `*` FROM Student;
  - Single-table query, so no cross product here
  - WHERE clause is optional
  - `*` is a short hand for “all columns”
Example: selection and projection

- Name of students under 18
  - SELECT name FROM Student WHERE age < 18;
- When was Lisa born?
  - SELECT 2009 - age
    FROM Student
    WHERE name = 'Lisa';
- SELECT list can contain expressions
  - Can also use built-in functions such as SUBSTR, ABS, etc.
- String literals (case sensitive) are enclosed in single quotes

Example: join

- SID’s and names of students taking courses with the word “Database” in their titles
  - SELECT Student.SID, Student.name
    FROM Student, Enroll, Course
    WHERE Student.SID = Enroll.SID
    AND Enroll.CID = Course.CID
    AND title LIKE '%Database%';
  - LIKE matches a string against a pattern
    - % matches any sequence of 0 or more characters
  - Okay to omit table_name in table_name.column_name if column_name is unique

Example: rename

- SID’s of all pairs of classmates
  - Relational algebra query:
    - SQL:
      SELECT e1.SID AS SID1, e2.SID AS SID2
      FROM Enroll AS e1, Enroll AS e2
      WHERE e1.CID = e2.CID
      AND e1.SID > e2.SID;
  - AS keyword is completely optional
A more complicated example

- Titles of all courses that Bart and Lisa are taking together

```
SELECT
FROM
WHERE
```

Tip: Write the FROM clause first, then WHERE, and then SELECT

Why SFW statements?

- Out of many possible ways of structuring SQL statements, why did the designers choose SELECT-FROM-WHERE?
  - A large number of queries can be written using only selection, projection, and cross product (or join)
  - Any query that uses only these operators can be written in a canonical form: \( \pi_L(\sigma_P(R_1 \times \ldots \times R_m)) \)
    - Example: \( \pi_{R.A,B} (R \bowtie_{P1} S \bowtie_{P2} (\pi_{T.C} T)) = \pi_{R.A,B,T.C} (\sigma_{P1 \bowtie_{P2} P3} (R \times S \times T)) \)
  - SELECT-FROM-WHERE captures this canonical form

Set versus bag semantics

- Set
  - No duplicates
  - Relational model and algebra use set semantics
- Bag
  - Duplicates allowed
  - Number of duplicates is significant
  - SQL uses bag semantics by default
Set versus bag example

\[ \pi_{SID} Enroll \]

<table>
<thead>
<tr>
<th>SID</th>
<th>CID</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>CPS118</td>
</tr>
<tr>
<td>142</td>
<td>CPS114</td>
</tr>
<tr>
<td>123</td>
<td>CPS118</td>
</tr>
<tr>
<td>857</td>
<td>CPS116</td>
</tr>
<tr>
<td>857</td>
<td>CPS116</td>
</tr>
<tr>
<td>456</td>
<td>CPS114</td>
</tr>
</tbody>
</table>

SELECT SID FROM Enroll;


A case for bag semantics

- Besides, SQL provides the option of set semantics with DISTINCT keyword

Forcing set semantics

- SID's of all pairs of classmates
  - SELECT e1.SID AS SID1, e2.SID AS SID2
    FROM Enroll AS e1, Enroll AS e2
    WHERE e1.CID = e2.CID
    AND e1.SID > e2.SID;

  - SELECT DISTINCT e1.SID AS SID1, e2.SID AS SID2
    ...
  - With DISTINCT, all duplicate (SID1, SID2) pairs are removed from the output
Operational semantics of SFW

- SELECT (DISTINCT) \( E_1, E_2, \ldots, E_n \)
  FROM \( R_1, R_2, \ldots, R_m \)
  WHERE condition;

- For each \( t_1 \) in \( R_1 \):
  For each \( t_2 \) in \( R_2 \): \ldots
  For each \( t_m \) in \( R_m \):
    If condition is true over \( t_1, t_2, \ldots, t_m \):
      Compute and output \( E_1, E_2, \ldots, E_n \) as a row

- If DISTINCT is present
  Eliminate duplicate rows in output

- \( t_1, t_2, \ldots, t_m \) are often called tuple variables

SQL set and bag operations

- UNION, EXCEPT, INTERSECT

  - Set semantics
    - Duplicates in input tables, if any, are first eliminated
    - Exactly like set \( \cup, \cap, \setminus \) in relational algebra
  - UNION ALL, EXCEPT ALL, INTERSECT ALL

  - Bag semantics
    - Think of each row as having an implicit count (the number of times it appears in the table)
    - Bag union: sum up the counts from two tables
    - Bag difference: proper-subtract the two counts
    - Bag intersection: take the minimum of the two counts

Examples of bag operations
Examples of set versus bag operations

- Enroll(SID, CID), ClubMember(club, SID)
  - (SELECT SID FROM ClubMember)
    - EXCEPT
      - (SELECT SID FROM Enroll);
  - (SELECT SID FROM ClubMember)
    - EXCEPT ALL
      - (SELECT SID FROM Enroll);

Summary of SQL features covered so far

- SELECT-FROM-WHERE statements (select-project-join queries)
- Set and bag operations

- Next: how to nest SQL queries

Table expression

- Use query result as a table
  - In set and bag operations, FROM clauses, etc.
  - A way to “nest” queries
- Example: names of students who are in more clubs than classes

  SELECT DISTINCT name
  FROM Student,
  (SELECT SID FROM ClubMember)
  EXCEPT ALL
  (SELECT SID FROM Enroll) AS S
  WHERE Student.SID = S.SID;
Scalar subqueries

- A query that returns a single row can be used as a value in WHERE, SELECT, etc.
- Example: students at the same age as Bart
  
  ```sql
  SELECT * FROM Student
  WHERE age = (SELECT age FROM Student WHERE name = 'Bart');
  ```
- Runtime error if subquery returns more than one row
  - Under what condition will this runtime error never occur?
- What if subquery returns no rows?
  - The return value is treated as a special value NULL, and the comparison fails
- Can be used in SELECT to compute a value for an output column

IN subqueries

- `x IN (subquery)` checks if `x` is in the result of subquery
- Example: students at the same age as (some) Bart
  
  ```sql
  SELECT * FROM Student
  WHERE age IN (SELECT age FROM Student WHERE name = 'Bart');
  ```

EXISTS subqueries

- `EXISTS (subquery)` checks if the result of subquery is non-empty
- Example: students at the same age as (some) Bart
  
  ```sql
  SELECT *
  FROM Student AS s
  WHERE EXISTS (SELECT * FROM Student
  WHERE name = 'Bart'
  AND age = s.age);
  ```
  - This happens to be a correlated subquery—a subquery that references tuple variables in surrounding queries
Operational semantics of subqueries

- SELECT *
  FROM Student AS s
  WHERE EXISTS (SELECT * FROM Student
                WHERE name = 'Bart'
                AND age = s.age);

- For each row s in Student
  - Evaluate the subquery with the appropriate value of s.age
  - If the result of the subquery is not empty, output s.*
- The DBMS query optimizer may choose to process the query in an equivalent, but more efficient way (example?)

Scoping rule of subqueries

- To find out which table a column belongs to
  - Start with the immediately surrounding query
  - If not found, look in the one surrounding that; repeat if necessary
- Use table_name.column_name notation and AS (renaming) to avoid confusion

Another example

SELECT * FROM Student s
WHERE EXISTS
  (SELECT * FROM Enroll e
   WHERE [SID] = s.SID
   AND EXISTS
     (SELECT * FROM Enroll
      WHERE [SID] = [s.SID]
      AND CID <> e.CID));
Quantified subqueries

- A quantified subquery can be used as a value in a `WHERE` condition.
- Universal quantification (for all):
  \[ \text{WHERE } x \text{ op } \text{ALL (subquery)} \ldots \]
  - True iff for all \( t \) in the result of subquery, \( x \text{ op } t \)
- Existential quantification (exists):
  \[ \text{WHERE } x \text{ op } \text{ANY (subquery)} \ldots \]
  - True iff there exists some \( t \) in the result of subquery such that \( x \text{ op } t \)

&bull; Beware
  - In common parlance, “any” and “all” seem to be synonyms
  - In SQL, ANY really means “some”

Examples of quantified subqueries

- Which students have the highest GPA?
  - `SELECT * FROM Student WHERE GPA >= ALL (SELECT GPA FROM Student);`
  - `SELECT * FROM Student WHERE NOT (GPA < ANY (SELECT GPA FROM Student));`
  - Use `NOT` to negate a condition

More ways of getting the highest GPA

- Which students have the highest GPA?
Summary of SQL features covered so far

- SELECT-FROM-WHERE statements
- Set and bag operations
- Table expressions, subqueries
  - Subqueries allow queries to be written in more declarative ways (recall the highest GPA query)
  - But they do not add much expressive power
    - Try translating other forms of subqueries into \( \text{NOT EXISTS} \), which in turn can be translated into join (and difference)

Next: aggregation and grouping

Aggregates

- Standard SQL aggregate functions: COUNT, SUM, AVG, MIN, MAX
- Example: number of students under 18, and their average GPA
  - \( \text{SELECT COUNT(*)}, \ \text{AVG(GPA)} \)
    - \( \text{FROM Student} \)
    - \( \text{WHERE age < 18;} \)
  - \( \text{COUNT(*)} \) counts the number of rows

Aggregates with \textit{DISTINCT}

- Example: How many students are taking classes?
  - \( \text{SELECT COUNT(DISTINCT SID)} \)
    - \( \text{FROM Enroll;} \)
    - is equivalent to:
GROUP BY

- \[
\text{SELECT} \ldots \text{FROM} \ldots \text{WHERE} \ldots \\
\text{GROUP BY} \text{list_of_columns};
\]

- Example: find the average GPA for each age group
  - \[
  \text{SELECT age, AVG(GPA)} \\
  \text{FROM Student} \\
  \text{GROUP BY} \text{age};
  \]

Operational semantics of GROUP BY

- \[
\text{SELECT} \ldots \text{FROM} \ldots \text{WHERE} \ldots \text{GROUP BY} \ldots;
\]
  - Compute FROM (\(\times\))
  - Compute WHERE (\(\sigma\))
  - Compute GROUP BY: group rows according to the values of GROUP BY columns
  - Compute SELECT for each group (\(\pi\))
  - For aggregation functions with DISTINCT inputs, first eliminate duplicates within the group
  - Number of groups = number of rows in the final output

Example of computing GROUP BY

- \[
\text{SELECT age, AVG(GPA)} \text{ FROM Student GROUP BY age;}
\]

<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>age</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td>10</td>
<td>2.3</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>3.1</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>8</td>
<td>4.3</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td>8</td>
<td>2.3</td>
</tr>
</tbody>
</table>

- Compute GROUP BY: group rows according to the values of GROUP BY columns
- Compute SELECT for each group
Aggregates with no GROUP BY

- An aggregate query with no GROUP BY clause represent a special case where all rows go into one group

SELECT AVG(GPA) FROM Student;

Group all rows into one group

Compute aggregate over the group

<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>age</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td>10</td>
<td>2.3</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>8</td>
<td>4.3</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>3.1</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td>8</td>
<td>2.3</td>
</tr>
</tbody>
</table>

Restriction on SELECT

- If a query uses aggregation/group by, then every column referenced in SELECT must be either
  - Aggregated, or
  - A GROUP BY column

Examples of invalid queries

- SELECT SID, age FROM Student GROUP BY age;
  - Recall there is one output row per group
  - There can be multiple SID values per group
- SELECT SID, MAX(GPA) FROM Student;
  - Recall there is only one group for an aggregate query with no GROUP BY clause
  - There can be multiple SID values
  - Wishful thinking (that the output SID value is the one associated with the highest GPA) does NOT work
  - Another way of writing the max GPA query?
HAVING

- Used to filter groups based on the group properties (e.g., aggregate values, GROUP BY column values)
- SELECT ... FROM ... WHERE ... GROUP BY ...
  HAVING condition;
  - Compute FROM (×)
  - Compute WHERE (σ)
  - Compute GROUP BY: group rows according to the values of GROUP BY columns
  - Compute HAVING (another σ over the groups)
  - Compute SELECT (π) for each group that passes HAVING

HAVING examples

- Find the average GPA for each age group over 10:
  SELECT age, AVG(GPA)
  FROM Student
  GROUP BY age
  HAVING age > 10;
  - Can be written using WHERE without table expressions
- List the average GPA for each age group with more than a hundred students:
  SELECT age, AVG(GPA)
  FROM Student
  GROUP BY age
  HAVING COUNT(*) > 100;
  - Can be written using WHERE and table expressions

Summary of SQL features covered so far

- SELECT-FROM-WHERE statements
- Set and bag operations
- Table expressions, subqueries
- Aggregation and grouping
  - More expressive power than relational algebra
- Next: ordering output rows
ORDER BY

- SELECT [DISTINCT] ...
  FROM ... WHERE ... GROUP BY ... HAVING ...
  ORDER BY output_column [ASC | DESC], ...;
- ASC = ascending, DESC = descending
- Operational semantics
  - After SELECT list has been computed and optional
duplicate elimination has been carried out,
sort the output according to ORDER BY specification

ORDER BY example

- List all students, sort them by GPA (descending)
  and name (ascending)
  - SELECT SID, name, age, GPA
    FROM Student
    ORDER BY GPA DESC, name;
  - ASC is the default option
  - Strictly speaking, only output columns can appear in
ORDER BY clause (although some DBMS support more)
  - Can use sequence numbers instead of names to refer to
output columns: ORDER BY 4 DESC, 2;

Summary of SQL features covered so far

- SELECT-FROM-WHERE statements
- Set and bag operations
- Table expressions, subqueries
- Aggregation and grouping
- Ordering

- Next: NULL's, outerjoins, data modification, constraints, …